

Information Technology Engineering

Class: G9-M2

Name: Ly Uttakra

Assignment: Database Design and Implementation

SQL queries with explanations:

1. Retrieve all students who enrolled in a specific course.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with databases like 'cg', 'database', 'DB1', 'ex', 'inventory', 'inventory1', 'om', 'sakila', 'SchoolDB', 'sys', and 'university'. The main area shows a query editor with the following SQL code:

```
1 •  SELECT s.student_id, s.first_name, s.last_name, s.email
2   FROM students s
3   JOIN enrollments e ON s.student_id = e.student_id
4   JOIN courses c ON e.course_id = c.course_id
5 WHERE c.course_code = 'CS101';
```

The results grid shows one row of data:

student_id	first_name	last_name	email
1	Alice	Johnson	alice@example.com

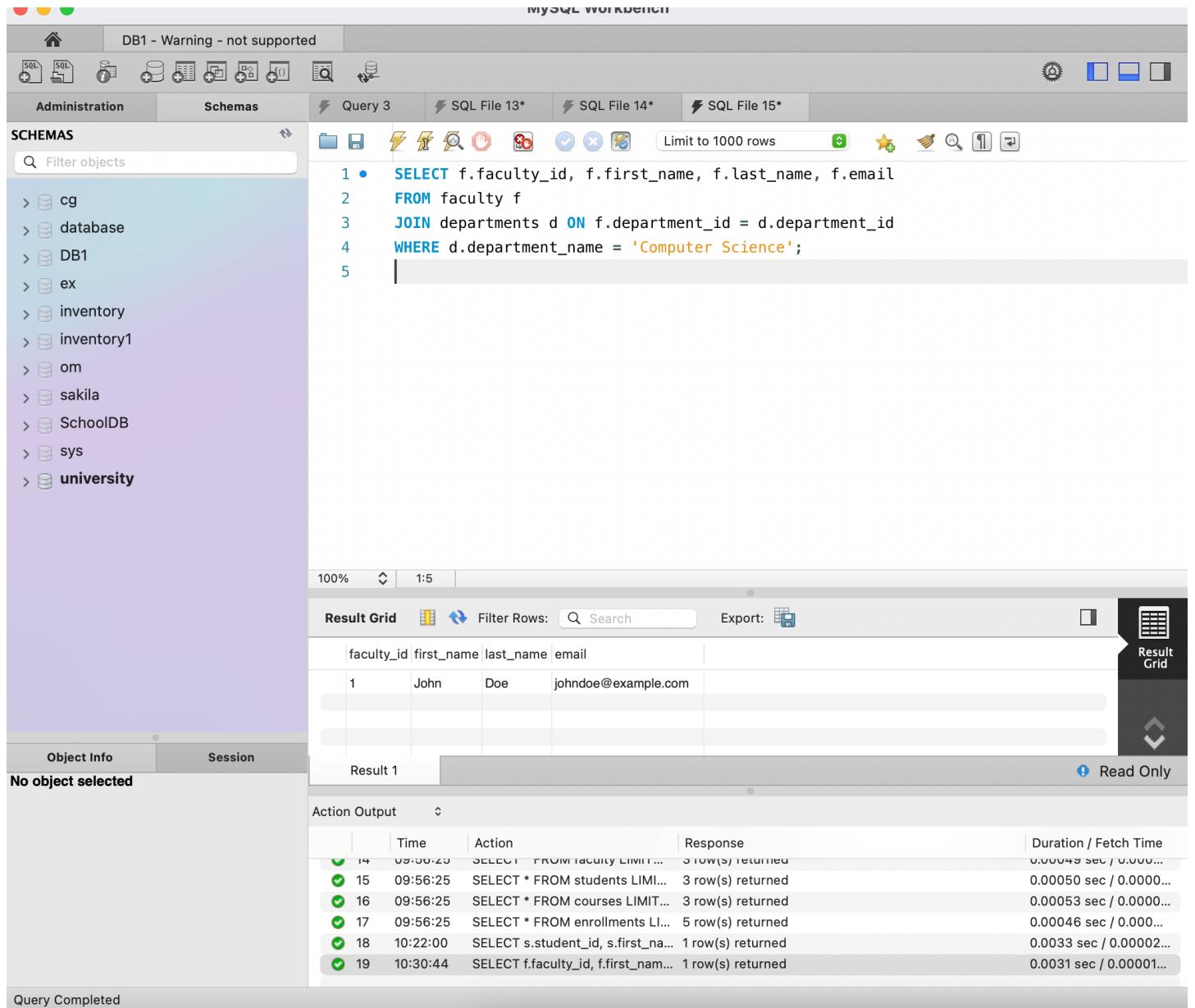
The 'Action Output' pane at the bottom lists the execution history:

Action	Time	Response	Duration / Fetch Time
13	09:56:25	SELECT * FROM departments L...	0.000000 sec / 0.000...
14	09:56:25	SELECT * FROM faculty LIMIT...	0.000049 sec / 0.000...
15	09:56:25	SELECT * FROM students LIMI...	0.000050 sec / 0.000...
16	09:56:25	SELECT * FROM courses LIMIT...	0.000053 sec / 0.000...
17	09:56:25	SELECT * FROM enrollments LI...	0.000046 sec / 0.000...
18	10:22:00	SELECT s.student_id, s.first_na...	0.0033 sec / 0.00002...

How it works:

- Merges the students, enrollments, and courses tables to identify students registered for a specific course.
- Applies a filter on course_code (substituting 'CS101' with the desired course).

2. Find all faculty members in a particular department.



The screenshot shows the MySQL Workbench interface. The left sidebar lists databases: cg, database, DB1, ex, inventory, inventory1, om, sakila, SchoolDB, sys, and university. The main area displays a SQL query in Query 3:

```
1 •  SELECT f.faculty_id, f.first_name, f.last_name, f.email
2   FROM faculty f
3   JOIN departments d ON f.department_id = d.department_id
4   WHERE d.department_name = 'Computer Science';
5
```

The Result Grid shows one row of data:

	faculty_id	first_name	last_name	email
1	1	John	Doe	johndoe@example.com

The Action Output section shows the history of actions:

Time	Action	Response	Duration / Fetch Time
09:50:25	SELECT * FROM faculty LIMIT...	3 row(s) returned	0.00049 sec / 0.000...
09:56:25	SELECT * FROM students LIMIT...	3 row(s) returned	0.00050 sec / 0.000...
09:56:25	SELECT * FROM courses LIMIT...	3 row(s) returned	0.00053 sec / 0.000...
09:56:25	SELECT * FROM enrollments LI...	5 row(s) returned	0.00046 sec / 0.000...
10:22:00	SELECT s.student_id, s.first_na...	1 row(s) returned	0.0033 sec / 0.00002...
10:30:44	SELECT f.faculty_id, f.first_na...	1 row(s) returned	0.0031 sec / 0.00001...

At the bottom, it says "Query Completed".

How it works:

- Combines the faculty and departments tables to associate faculty members with their respective departments.
- Applies a filter on the department name (replace 'Computer Science' with the desired department).

3. List all courses a particular student is enrolled in.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree, which includes 'DB1', 'cg', 'database', 'ex', 'inventory', 'inventory1', 'om', 'sakila', 'SchoolDB', 'sys', and 'university'. The main area contains a SQL editor with the following query:

```

1 •   SELECT c.course_code, c.course_name
2     FROM courses c
3   JOIN enrollments e ON c.course_id = e.course_id
4   JOIN students s ON e.student_id = s.student_id
5 WHERE s.email = 'alice@example.com';

```

The results grid shows two rows of data:

course_code	course_name
CS101	Introduction to Programming
MATH201	Calculus I

The 'Action Output' section at the bottom lists the actions taken by the database during the query execution:

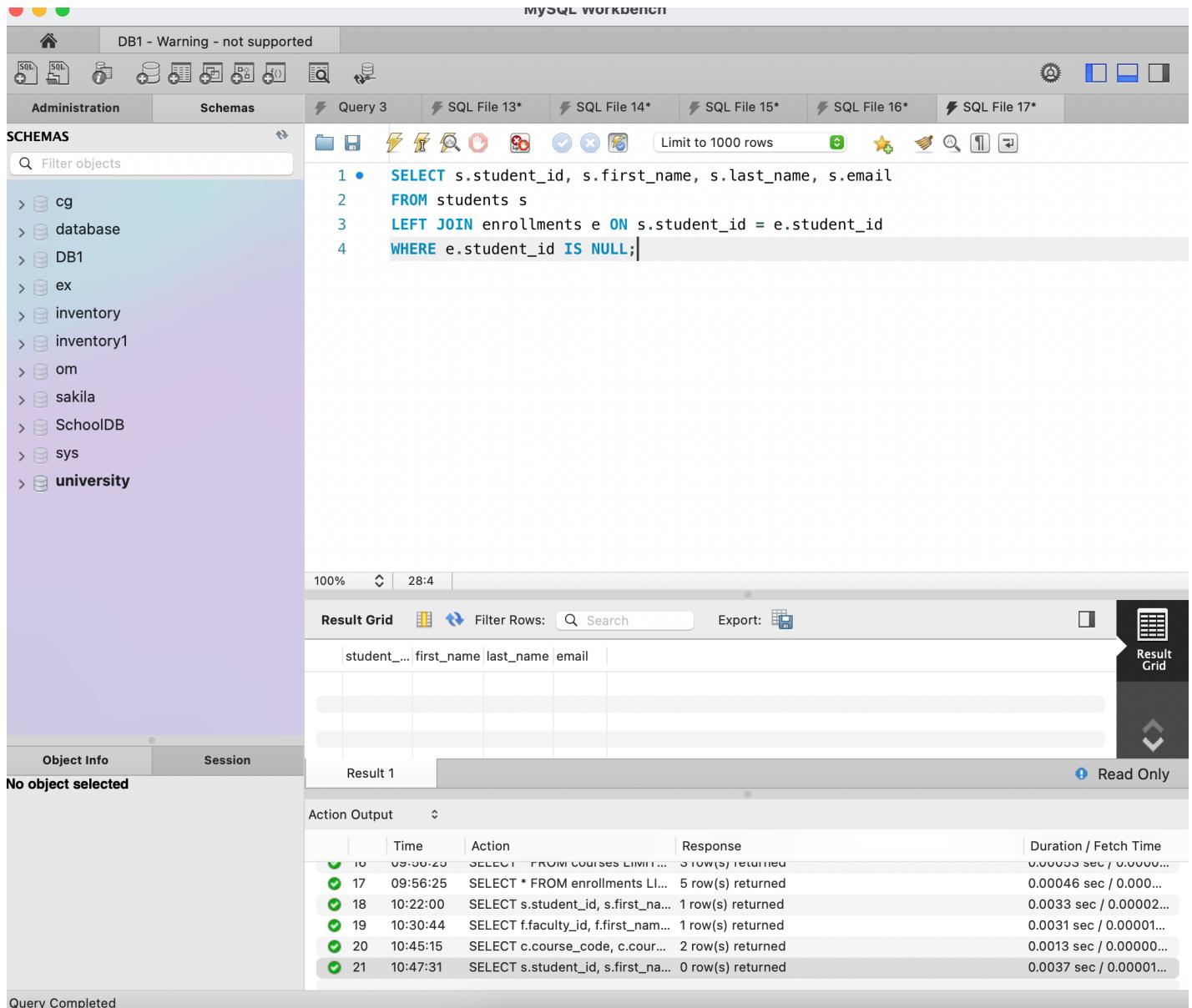
Action	Time	Response	Duration / Fetch Time
SELECT * FROM students LIMIT...	09:56:23	3 row(s) returned	0.00000 sec / 0.0000...
SELECT * FROM courses LIMIT...	09:56:25	3 row(s) returned	0.00053 sec / 0.0000...
SELECT * FROM enrollments LI...	09:56:25	5 row(s) returned	0.00046 sec / 0.0000...
SELECT s.student_id, s.first_na...	10:22:00	1 row(s) returned	0.0033 sec / 0.00002...
SELECT f.faculty_id, f.first_nam...	10:30:44	1 row(s) returned	0.0031 sec / 0.00001...
SELECT c.course_code, c.cour...	10:45:15	2 row(s) returned	0.0013 sec / 0.00000...

The status bar at the bottom indicates 'Query Completed'.

How it works:

- Joins the students, enrollments, and courses tables to retrieve courses linked to a specific student.
- Filters the results using the student's email (replace 'alice@example.com' with any student's email).

4. Retrieve students who have not enrolled in any course.



The screenshot shows the MySQL Workbench interface. The top menu bar displays "MySQL WORKBENCH". The left sidebar lists databases: cg, database, DB1, ex, inventory, inventory1, om, sakila, SchoolDB, sys, and university. The main area contains a SQL editor with the following query:

```

1 •  SELECT s.student_id, s.first_name, s.last_name, s.email
2   FROM students s
3   LEFT JOIN enrollments e ON s.student_id = e.student_id
4 WHERE e.student_id IS NULL;
    
```

The results grid below shows the output of the query. The "Result Grid" tab is selected, displaying columns: student_id, first_name, last_name, and email. The "Action Output" tab shows the history of actions taken during the session, including the execution of the current query and previous statements.

Action	Time	Response	Duration / Fetch Time
16 10:30:23	SELECT * FROM courses LIMIT ...	3 row(s) returned	0.00003 sec / 0.0000...
17 09:56:25	SELECT * FROM enrollments LI...	5 row(s) returned	0.00046 sec / 0.000...
18 10:22:00	SELECT s.student_id, s.first_na...	1 row(s) returned	0.0033 sec / 0.00002...
19 10:30:44	SELECT f.faculty_id, f.first_nam...	1 row(s) returned	0.0031 sec / 0.00001...
20 10:45:15	SELECT c.course_code, c.cour...	2 row(s) returned	0.0013 sec / 0.00000...
21 10:47:31	SELECT s.student_id, s.first_na...	0 row(s) returned	0.0037 sec / 0.00001...

How it works:

- Performs a LEFT JOIN between students and enrollments, ensuring all students remain in the result.
- Excludes students who have at least one matching enrollment record.
- A NULL value in e.student_id indicates that the student has no enrollments.

5. Find the average grade of students in a specific course.

MySQL Workbench screenshot showing the execution of a SQL query to find the average GPA of students in the CS101 course.

```

    WHEN 'B+' THEN 3.3
    WHEN 'B' THEN 3.0
    WHEN 'B-' THEN 2.7
    WHEN 'C+' THEN 2.3
    WHEN 'C' THEN 2.0
    WHEN 'C-' THEN 1.7
    WHEN 'D+' THEN 1.3
    WHEN 'D' THEN 1.0
    WHEN 'F' THEN 0.0
    ELSE NULL
END
ELSE NULL
END) AS average_gpa
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
WHERE c.course_code = 'CS101';

```

The Result Grid shows the output:

course_code	course_name	average_gpa
CS101	Introduction to Programming	4.00000

Action Output table:

Time	Action	Response	Duration / Fetch Time
09:50:20	SELECT * FROM enrollments	5 row(s) returned	0.00040 sec / 0.0000...
10:22:00	SELECT s.student_id, s.first_na...	1 row(s) returned	0.0033 sec / 0.00002...
10:30:44	SELECT f.faculty_id, f.first_nam...	1 row(s) returned	0.0031 sec / 0.00001...
10:45:15	SELECT c.course_code, c.cour...	2 row(s) returned	0.0013 sec / 0.00000...
10:47:31	SELECT s.student_id, s.first_na...	0 row(s) returned	0.0037 sec / 0.00001...
10:51:42	SELECT c.course_code, c.cour...	1 row(s) returned	0.012 sec / 0.000012...

How it works:

- Joins the courses and enrollments tables to retrieve student grades for a specific course.
- Maps letter grades (A, B+, etc.) to their corresponding GPA values (e.g., A+ = 4.3, F = 0).
- Calculates the average GPA of students enrolled in the course.

Bonus:

I. Implement a trigger to update a student's GPA when a grade is updated.

a. Step 1 : Add a gpa Column to students Table By

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the schema is set to 'DB1 - Warning - not supported'. The left sidebar lists various databases: cg, database, DB1, ex, inventory, inventory1, om, sakila, SchoolDB, sys, and university. The main query editor window contains the SQL command: `ALTER TABLE students ADD COLUMN gpa DECIMAL(3,2) DEFAULT NULL;`. Below the query editor, the 'Session' tab is selected in the bottom navigation bar. The 'Action Output' section displays the execution history of the session, showing the following log entries:

Time	Action	Response	Duration / Fetch Time
10:22:00	SELECT s.student_id, s.first_name...	1 row(s) returned	0.0003 sec / 0.00002...
19 10:30:44	SELECT f.faculty_id, f.first_name...	1 row(s) returned	0.0031 sec / 0.00001...
20 10:45:15	SELECT c.course_code, c.cour...	2 row(s) returned	0.0013 sec / 0.00000...
21 10:47:31	SELECT s.student_id, s.first_na...	0 row(s) returned	0.0037 sec / 0.00001...
22 10:51:42	SELECT c.course_code, c.cour...	1 row(s) returned	0.012 sec / 0.000012...
23 10:54:44	ALTER TABLE students ADD C...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.020 sec	

At the bottom of the interface, the message 'Query Completed' is displayed.

b. Step 2: Create the Trigger

The screenshot shows the MySQL Workbench interface with the following details:

- Title Bar:** MySQL Workbench - DB1 - Warning - not supported
- Schemas Tab:** Shows the current schema is DB1, with objects like cg, database, DB1, ex, inventory, inventory1, om, sakila, SchoolDB, sys, and university.
- SQL Editor:** Contains the SQL code for creating a trigger:

```

    BEGIN
        UPDATE students
        SET gpa = (
            SELECT AVG(CASE
                WHEN e.grade = 'A+' THEN 4.3
                WHEN e.grade = 'A' THEN 4.0
                WHEN e.grade = 'A-' THEN 3.7
                WHEN e.grade = 'B+' THEN 3.3
                WHEN e.grade = 'B' THEN 3.0
                WHEN e.grade = 'B-' THEN 2.7
                WHEN e.grade = 'C+' THEN 2.3
                WHEN e.grade = 'C' THEN 2.0
                WHEN e.grade = 'C-' THEN 1.7
                WHEN e.grade = 'D+' THEN 1.3
                WHEN e.grade = 'D' THEN 1.0
                WHEN e.grade = 'F' THEN 0.0
            ELSE NULL
            END)
            FROM enrollments e
            WHERE e.student_id = NEW.student_id
        )
        WHERE student_id = NEW.student_id;
    END $$
```
- Object Info Tab:** No object selected
- Action Output Tab:** Shows the history of actions with their times, responses, and durations:

Action	Time	Response	Duration / Fetch Time
19	10:39:44	SELECT t1.student_id, t1.first_name...	0.0001 sec / 0.00001...
20	10:45:15	SELECT c.course_code, c.cour...	0.0013 sec / 0.00000...
21	10:47:31	SELECT s.student_id, s.first_na...	0.0037 sec / 0.00001...
22	10:51:42	SELECT c.course_code, c.cour...	0.012 sec / 0.000012...
23	10:54:44	ALTER TABLE students ADD C...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.020 sec
24	11:11:19	CREATE TRIGGER update_gpa...	0 row(s) affected 0.013 sec
- Status Bar:** Query Completed

How it works:

- Transforms letter grades (A, B+, etc.) into GPA values on a 4.0 scale.
- Computes a student's average GPA from all their grades.
- Updates the gpa column in the students table with the calculated value.

II. Design a stored procedure to enroll a student in a course.

The screenshot shows the MySQL Workbench interface with the following details:

- Left Panel (Schemas):** Shows the database schema with various databases listed: cg, database, DB1, ex, inventory, inventory1, om, sakila, SchoolDB, sys, and university.
- Central Panel (Query Editor):** Displays the SQL code for the stored procedure:

```
DELIMITER $$  
CREATE PROCEDURE enroll_student(IN p_student_id INT, IN p_course_id INT)  
BEGIN  
    DECLARE enrollment_exists INT;  
  
    -- Check if the student is already enrolled in the course  
    SELECT COUNT(*) INTO enrollment_exists  
    FROM enrollments  
    WHERE student_id = p_student_id AND course_id = p_course_id;  
  
    -- If student is not enrolled, insert new enrollment  
    IF enrollment_exists = 0 THEN  
        INSERT INTO enrollments (student_id, course_id, enrollment_date)  
        VALUES (p_student_id, p_course_id, CURDATE());  
        SELECT CONCAT('Student ', p_student_id, ' successfully enrolled in Course ', p_course_id) AS Message;  
    ELSE  
        SELECT 'Student is already enrolled in this course!' AS Message;  
    END IF;  
END $$
```
- Bottom Panel (Session):** Shows the execution results:

Action Output	Time	Action	Response	Duration / Fetch Time
1	19:35:02	CREATE PROCEDURE enroll_st...	0 row(s) affected	0.00083 sec

How it works:

- Verifies whether the student is already enrolled in the course.
- If not enrolled, adds a new record to the enrollments table.
- Returns a success message or an error message if the student is already registered.

Testing the Stored Procedure

MySQL Workbench

DB1 - Warning - not supported

Administration Schemas

SCHEMAS

Filter objects

> cg
> database
> DB1
> ex
> inventory
> inventory1
> om
> sakila
> SchoolDB
> sys
> university

Query 3 SQL File 13* SQL File 14* SQL File 15* SQL File 16* SQL File 17* >

1 • CALL enroll_student(1, 2);

Result Grid Filter Rows: Search Export:

Message

Student is already enrolled in this course!

Result 1 Read Only

Action Output

	Time	Action	Response	Duration / Fetch Time
1	19:38:55	CALL enroll_student(1, 2)	1 row(s) returned	0.012 sec / 0.000016...

Query Completed Microsoft A

Result Grid Form Editor Field Types